Landscape Access Model (LAM) for the Park Units in the Northern Colorado Plateau Network

Steven L. Garman
Ecologist, Northern Colorado Plateau Network
National Park Service
2282 SW Resource Blvd.
Moab, UT 84532

Version 1.0 (6/21/05)

Table of Contents

Figures

Introduction

The Northern Colorado Plateau Network (NCPN) Inventory and Monitoring program is developing survey sampling procedures to monitor ecological attributes in a 16-park network. A critical first-step in crafting a monitoring sampling design is to determine the portion of a park landscape that is accessible on foot, and the associated travel costs (i.e., minimum hiking distance). This information is important for two reasons. First, steep and rugged terrain in the NCPN severely limits access to many canyon bottoms and mesa tops. Budgetary and safety issues prohibit the use of helicopter transport and mountaineering methods, such as rock climbing, to reach these isolated sites. Areas that can not be accessed safely on foot must be eliminated from consideration when selecting sampling plots. Second, the larger parks in the NCPN have considerable amount of backcountry. Sampling front- and back-country equally is cost prohibitive. Identification of inaccessible areas and estimates of travel costs to accessible areas on a park landscape promotes efficient allocation and delineation of sampling locations for monitoring.

The NCPN developed a customized model that relies on GIS-derived raster layers of topography and transportation networks to determine accessibility and minimum hiking distance from roads to every landscape cell. This model, called Landscape Access Model (LAM), is coded in C, consists of two separate modules, and runs on standard PC computers. The model is fairly simple, and was designed as a starting point for accessibility and travel-cost analyses. Evaluation of initial estimates, site-specific issues and considerations, etc., likely will necessitate fine-tuning of underlying assumptions and algorithms. Enhancements are easily incorporated, but require programming expertise. It is the intent to eventually translate the underlying algorithms to a user-friendly GIS program.

This document describes the basic features of the LAM system, including input requirements, calculations, and output, and the syntax for running the two modules.

Data Requirements

LAM operates on grid-based data layers. All input grids (s.l. layers) must have the same grain size. A 10-m spatial grain typically is used for the NCPN. This grain size is a reasonable compromise between accuracy of distance values and the size of input files. Finer grain sizes substantially increases file size, and unnecessarily adds to processing time. An additional requirement is that all layers must be of the same extent; that is, they must have the same number of rows and columns. And, of course, all layers must be geo-registered.

The original version of LAM required flat-binary file formats. A known problem with a C I/O function, however, limited the size of maps that could be read and written. To avoid this problem, the current version of LAM reads and writes ASCII formatted grids. The ASCII format is the same used in ArcGIS.

Input Layers

The required data layers are listed below along with required values. Background is other than the indicated values:

> parkboundary_layer – designates park area ($\leq$0 outside of park, >0 inside park)
> slope_layer – percent slope ($\geq$ 0)
> aspect_layer – degrees (-1 for flat; else 0-360).
> trails_layer – trails (>0 indicates a trail)
> roads_layer – roads (>0 indicates a road)
> (trails and roads can be combined into one layer, where trails are codified as 1 and roads are codified as >1).

All input and output files are in ArcGIS ASCII format. ArcGIS files are converted from a coverage to ASCII with the gridascii Arc command. LAM results can be converted to ArcGIS coverages with the asciigrid Arc command. Input layers can be any data type. However, LAM internally converts all input files to short integers (i.e., 16-bit number), and outputs map data as integers (i.e., 32-bit number). The internal use of short-ints limits the magnitude of a distance value (see below), but is efficient in terms of memory needs.

Memory Requirements

All spatial data layers are read into memory at program initiation. Memory-resident matrices used to store the data layers are responsible for much of the memory needs. Additionally, there are a series of matrices that record interim calculations, and these also are dynamically allocated at program initiation. Memory is allocated based on the number of rows and columns of the data layers which are supplied by the user. Thus, any sized ‘map’ can be processed without code changes. However, standard PC computers and operating systems are limited in the amount of memory that can be allocated per process. This limit is currently 3 GB on MS-Windows and MS-XP operating systems. Memory-allocation checks are embedded in the LAM modules. If memory limits are exceeded, an error message is displayed and processing immediately terminates.

Memory requirements for each module can be estimated with the following formula:

```
trail_roads_dista.exe         GB   =  (  ( 14  *  r  *  c ) / 1024000000.0 ) + 0.3
travel_costsa.exe             GB   =  (  ( 16  *  r  *  c ) / 1024000000.0 ) + 0.3
```

where;

r is no. of rows,
c is no. of columns,
1024000000.0 converts from bytes to GB,
0.3 is an estimate of memory needs for other code components,
GB is the total memory requirement in GigaBytes.

Overview of Cell-based Distance Calculations

LAM ultimately derives the minimum hiking distance from a road to every landscape cell that can be accessed on foot. This process involves deriving and evaluating distance values between neighboring cells. Distance is accumulated across every cell of the landscape starting at a road. Distance is calculated between the centroid of a focal cell and the centroid of each of its eight (adjacent and diagonal) neighboring cells. A focal cell is simply the cell for which distance is being calculated. Distance is adjusted by the slope and aspect of a cell, and the direction of travel across a cell (i.e., direction from a neighbor to the focal cell). Distance from the centroid of a neighboring cell to the edge with the focal cell, and from this edge to the centroid of the focal cell are derived separately, then summed to give the neighbor-focal cell distance value. Cell-based distance calculations used in both of the LAM modules are summarized below.

1) A required input layer is percent slope, which internally is converted to fractional slope. Fractional slope is used to derive a slope distance for a cell (Fig. 1). Slope distance is the actual surface distance represented by the cell, at least along the axis parallel to the slope (i.e., perpendicular to the contour). This slope distance only is used as an approximation for road cells, as explained below.

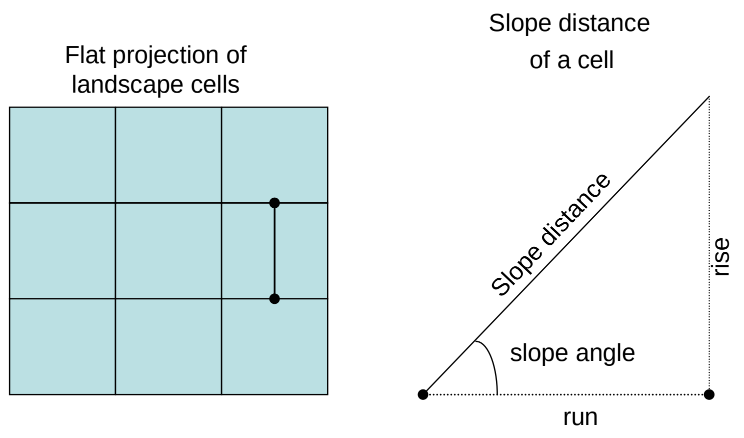
2) All distance estimates between pairs of cells are based on the fractional slope and flat-surface distance in the direction of travel. The direction of travel is essentially the azimuth from a neighbor to the focal cell. This azimuth is referred to as the travel angle. Each neighbor has an implicit travel angle (Fig. 2). Travel angle is used to determine the fractional slope of travel and the flat-surface travel distance.

> <u>Fractional Slope of Travel</u>. This is the slope experienced when traversing a cell at a given travel angle. As the deviation between the travel angle and aspect approaches 90 degrees, the fractional slope of the cell decreases. For instance, when traversing a cell parallel to the slope, the full fractional slope is experienced, and the actual hiking distance is equal to the slope distance. However, when traversing perpendicular to the slope, there is no slope adjustment to distance;

distance traveled is simply the edge-length of square cell. The derivation of the fractional slope given a travel angle is illustrated in Fig. 3. For simplicity, the deviation between the aspect and the direction of travel is scaled between 0 and 90 degrees. The maximum fractional slope is adjusted based on one minus the ratio of the deviation and 90 degrees. This assumes a linear decrease in the fractional slope between 0 (i.e., walking parallel to the slope) and 90 degrees (i.e., walking perpendicular to the slope).

Flat-surface Travel Distance. This is the flat-surface distance from the centroid to the edge of a cell for a given travel angle. For non-isodiametric cells (e.g., square cells), diagonal distance differs from vertical and horizontal distance. The flat-surface distance takes into account the edge-length of a cell and the angle at which a cell is traversed (Fig. 4). Given the raster design and the scaling of angles between 0 and 90 degrees, three travel angles are possible; 0, 45, and 90 degrees. A look-up table is used to derive the Tangent of the travel angle, which in turn, is used to derive the vertical component of the flat-surface distance. Euclidean geometry is employed to derive the flat-surface travel distance.

3) The slope-adjusted travel distance is the actual distance experienced at a given travel angle. The flat-surface travel distance is the horizontal component of the slope-adjusted travel distance (Fig. 5). This component times the fractional slope of travel is the vertical component of the slope-adjusted distance. Euclidean geometry is employed to derive the final, slope-adjusted distance.



rise = slope * run (1)

slope_dist = Sqrt $(rise^2 + run^2)$ (2)

where;

run is edge length of a square cell (m) and is the horizontal component of slope distance,
slope is fractional slope [Tan(slope_angle)],
rise is the vertical component of slope distance (m),
slope_dist is the slope distance (m).

Figure 1. Derivation of slope distance for a landscape cell

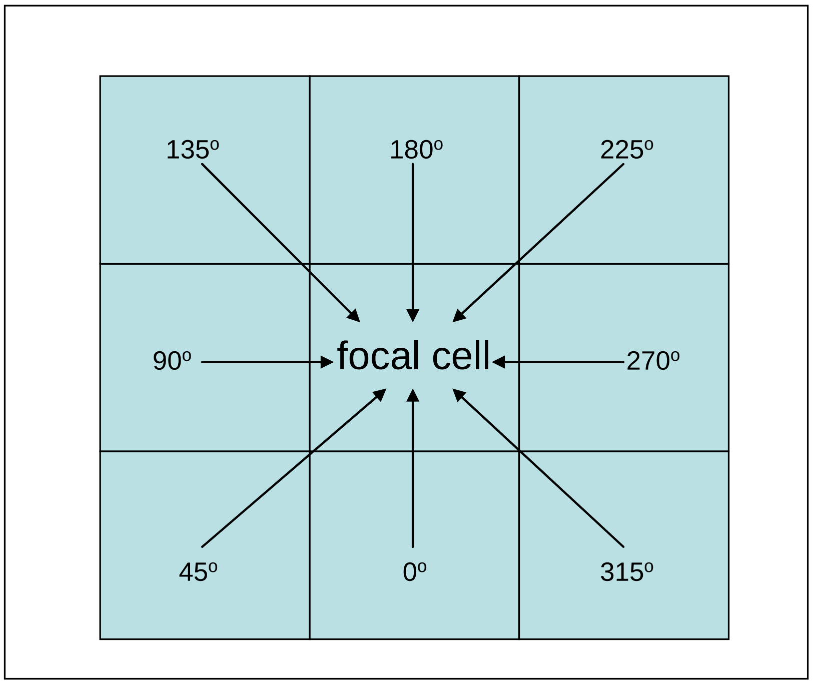
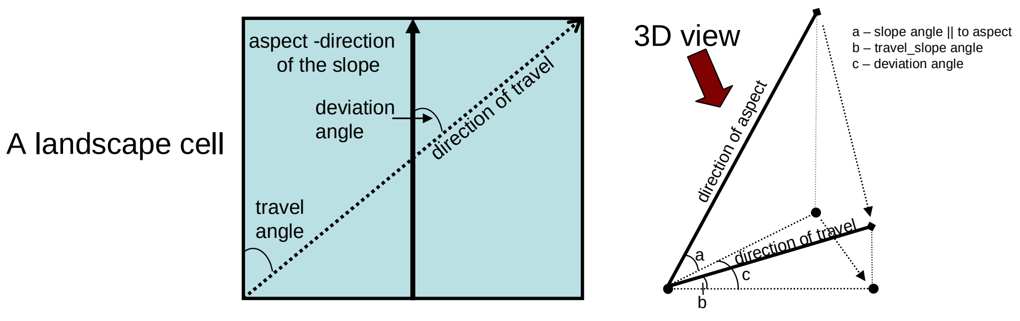


Figure 2. Implicit travel angle for the eight neighbors of a focal cell.

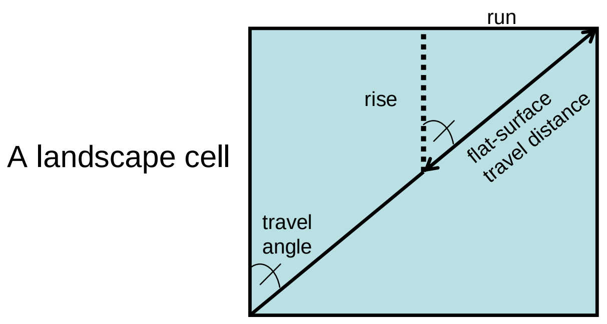


dev_angle = travel_angle - aspect (1)
slope_reduc = 1.0 – ((Acos (Fabs(Cos(dev_angle))))/90.0) (2)
travel_slope = slope_reduc * slope (3)

where;

aspect is the azimuth of the slope (degrees),
travel_angle is the azimuth of travel (degrees),
dev_angle is the difference between aspect and travel angle (degrees),
Fabs is an intrinsic function that derives the absolute value as a floating point number,
Acos scales the absolute value of the Cosine from 0 to 90 degrees,
slope_reduc is the slope reduction proportion,
slope is the fractional slope parallel to the aspect,
travel_slope is the fractional slope that is experienced when traversing the cell at travel_angle.

Figure 3. Derivation of fractional slope for a given direction of travel across a landscape cell.

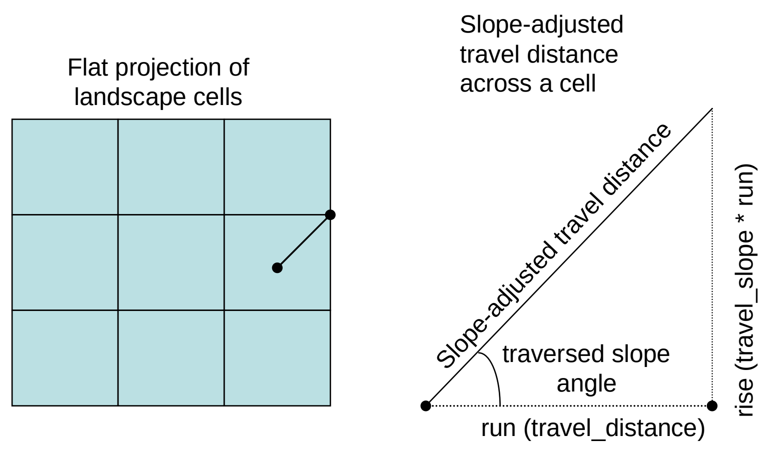


rise = Tan (travel_angle) * run (1)
travel_distance = Sqrt (rise^2 + run^2) (2)

where;

run is one-half the edge length of a square cell (m), and is the horizontal component of travel distance,
travel_angle is the direction of travel relative to north (degrees), and is scaled between
0 and 90 degrees,
Tan (travel_angle) is derived from a look-up table populated for 0, 45, and 90 degrees,
rise is the vertical component (m) of travel distance,
travel_distance is the flat-surface distance (m) from the centroid to the cell edge given the travel angle.

Figure 4. Derivation of flat-surface distance from the centroid to an edge of a landscape cell given a travel angle.


Flat projection of landscape cells
Slope-adjusted travel distance across a cell
Slope-adjusted travel distance
traversed slope angle
run (travel_distance)
rise (travel_slope * run)

rise = travel_slope * run (1)

slope_adj_dist = Sqrt (rise^2 + run^2) (2)

where;

run is the flat-surface travel distance (m) given the travel angle (see Fig. 4), and is the horizontal component of the slope-adjusted travel distance,

travel_slope is the fractional slope given the travel angle (see Fig. 3),

rise is the vertical component of slope-adjusted distance (m),

slope_adj_dist is the slope-adjusted travel distance (m).

Figure 5. Derivation of slope-adjusted travel distance.

Processing Procedures

LAM is comprised of two modules. The first is called trail_roads_dista, and derives the minimum hiking distance along trails starting from a trail-road intersection. This information is used in the second module. The second module is called travel_costsa, and determines areas that are inaccessible on foot, and derives a minimum cumulative distance value to the remaining accessible landscape cells. Separate modules evolved from earlier efforts with the LAM system. Often it was instructive to evaluate trail-distance estimates, which are relatively quick to produce, prior to the more time-consuming derivation of accessibility and landscape-cell distances.

Deriving trail distances from roads – trail_roads_dista.exe

The module, trail_roads_dista.exe, derives the minimum, slope-adjusted travel distance from roads to every trail cell. Distance calculations begin at trail-road intersections. These intersections can be outside as well as inside the park. However, if a trail extends inside the park boundary but the input data layers do not indicate an intersection with a road, distance values are not generated for the trail. Thus, the spatial extent of the input data layers must be large enough to include all possible trail-road intersections for trails on park lands. The slope-adjusted travel distance value for trail cells overlapping or adjacent to road cells is stored as a cumulative distance value (CDV) in an internal matrix termed the CDV matrix. Only the minimum CDV is stored.

After identifying trail-road intersections, the next step is the derivation of the CDV for every trail cell. Essentially, this involves accumulating distance along a trail from a trail-road intersection. The cumulative distance for a trail cell (i.e., the focal cell) is based on the CDV of a neighbor cell plus the distance between the neighbor and the focal cell. When a trail cell is encountered, neighbors with a CDV are determined. For each neighbor with a CDV, the sum of this CDV and the distance between the focal cell and the neighbor cell are derived. This newly derived CDV is compared with the stored CDV of the focal cell. If the focal cell lacks a CDV, the newly derived CDV is stored for the focal cell. Otherwise, the CDV for the focal cell is set to the minimum of the currently stored CDV and the newly derived CDV. Internally, a CDV is limited to the size of a signed, short integer. If a calculated CDV is >32768, the CDV is re-assigned to 32768, a warning message appears on the computer screen, but processing continues. This maximum distance value has been an issue for only one NCPN park. Program enhancements are being considered to increase this maximum distance value.

An iterative scanning procedure is used to derive a minimum CDV for all trail cells. This procedure essentially scans the landscape and assesses the minimum CDV of each encountered trail cell. The landscape in this module is the full spatial extent of the input data layers. Scanning commences at the top of the landscape and moves from left to right sequentially down the data layer. This scanning process is repeated starting at the bottom of the landscape and moving left to right up to the top. Scanning passes also occur from bottom to top, left to right; and bottom to top, right to left. The directionality of processing influences CDV estimates. The multiple-direction scanning procedure is a simple way to mix-up the directionality of calculations. The four scanning passes constitute one iteration. The program internally records the number of times a minimum CDV is replaced each iteration. When no values are replaced in an iteration, the minimum-distance surface has been derived, and the program outputs the final grid of minimum CDVs and terminates.

The output grid is formatted as an ArcGIS ASCII file, and contains the minimum CDV in meters for every trail cell in the input landscape. This output is used in the next module. The current design of the subsequent module requires a distance value for every road cell. To accommodate this need, one-half of the slope distance (see Fig. 1) is stored for every road cell. This value is not used in deriving cumulative distances in subsequent processing, but only indicates the presence of a road. This requirement has an historical context, and will be modified in future enhancements. Background (non-road and non-trial cells) in the output grid is set to -9999.

Program Syntax. To operate trail_roads_dista, you must store the executable as trail_roads_dista.exe. To run this module, enter the following:

```
trail_roads_dista <rows> <columns> <spatial_grain_m> <no. of iterations> <slope_layer>
<aspect_layer>  <trail_layer> <road_layer> <output_file>
```

where;

<rows> is the number of rows in the data layers,
columns> is the number of columns in the data layers,
<spatial_grain_m> is the spatial grain of the data layers in meters,
<no. of iterations> is the maximum number of internal iterations (passes) through the landscape – this is simply a safe-guard. Use 10000.
<slope_layer> is the name of the percent-slope layer,
<aspect_layer> is the name of the aspect layer,
<trail_layer> is the name of the trail layer,
<road_layer> is the name of the road layer,
<output_file> is the file name for the resulting trail-distance grid formatted as an ArcGIS ASCII file.

If trails and roads are combined in one layer, the combined layer must be entered in place of trail_layer, and the road_layer entry should be replaced with the word none; e.g.,

```
trail_roads_dista 1000 1514 10 10000 slope.asc aspect.asc trailroads.asc none
trail_dist.asc
```

Specifying the number of rows and columns, and grain size is admittedly redundant given that this information is available in the ASCII format of the input data layers. This requirement is a carry-over from the LAM modules that relied on binary-formatted layers, and will be changed in future versions.

The program trail_roads_dista.exe can reside anywhere on your computer, but an explicit path to the directory containing the program must be inserted via the PATH command to execute the program from any directory. Alternatively, the path can be specified in the command-line syntax. E.g., h:\access\trail_roads_dista <> <> … .

Determining accessibility and minimum hiking distance – travel_costsa.exe

The module, travel_costsa.exe, determines areas that are too steep for foot travel, areas that are surrounded by steep slopes and thus can not be accessed (e.g., mesas, canyon bottoms), and the minimum slope-adjusted travel distance (i.e., hiking distance) from a road to all other landscape cells. Cells with percent slope equal to or greater than a user-provided slope-mask value are codified as “too steep”, and do not accumulate distance values (i.e., a coded value [-1] is stored in the internal CDV matrix). Areas surrounded by steep slopes are identified and codified as isolated (-2), and also do no accumulate distance values. An exception is where "isolated" areas can be reached by existing trails. An assumption is that current trails over steep terrain afford reasonably easy and safe hiking access to these otherwise isolated areas. The trail-distance layer, which is the output from trail_roads_dista.exe, is employed in calculating cumulative distance values (CDV) for trail-accessible areas surrounded by steep slopes.

The basic processing methods of this module are similar to those of trail_roads_dista.exe, with additional steps and assumptions. First, all cells with percent slope equal to or greater than the slope-mask value are coded as a -1 (i.e., too steep). Then CDVs for cells adjacent to roads are derived. The iterative, multi-directional scanning procedure described above is used to derive a minimum CDV for every landscape cell not coded as a -1. In this module, distance accumulates from a road to other landscape cells one cell at a time, and is not solely based on trails. This produces what can be thought of as a cross-country distance; that is, the shortest distance across the landscape starting from a road. Currently, it is assumed that vegetation does not inhibit cross-country travel. In the dryland ecosystems of the NCPN, this is a logical assumption. However, in forested or woodland systems this assumption may not hold, and modeled results may under-estimate travel distance in heavily vegetated areas. Distance values in the trail-distance layer are used in conjunction with the cross-country calculations in deriving a minimum CDV for a landscape cell. For areas surrounded by steep slopes (s.l., isolated areas) but accessible by an existing trail, the trail-distance layer is the primary source for initiating calculations of cumulative distance. When minimum CDVs are achieved, the internal CDV storage for areas surrounded by steep slopes without trail access will contain a null value, and is codified as isolated (-2) prior to writing the minimum CDV grid to disk.

Values in the trail-distance layer can be internally static or dynamic. The trail-distance layer, output by trail_roads_dista.exe, contains the accumulated slope-adjusted travel distance along a trail from a road-trail intersection. There are instances, however, where a trail could be 'shortened' by cross-country travel. The **dynamic** option allows the trail-distance values to be internally updated with the minimum of cross-country CDVs and the original CDVs produced by trail_roads_dista.exe. Programmatically, the dynamic option integrates the CDVs of the trail-distance layer with the internally created and stored CDV matrix at program initiation. This initializes the landscape with CDVs for cells that overlap trails (and roads). These initial CDVs can change (i.e., decrease) as a result of the cross-country distance calculations. This option does set CDV values for trail cells on steep slopes, but they are codified to a -1 (too steep) prior to outputting the results. So even though steep-slope trail cells may function as a travel corridor, they are never classified as accessible landscape cells. The **static** option retains the CDVs values in the input trail-distance layer by not integrating distance values of trails with the internal distance matrix, and by using this layer as a "read-only" source of distance values. The noticeable difference between the static and dynamic options is in the CDVs of trail-accessible areas surrounded by steep slopes, with somewhat shorter distances produced with the dynamic option. Where off-trail travel is discouraged, the static version may be more desirable. Where off-trail travel isn't problematic, the dynamic option may be more representative. The user can choose to use either internally static or dynamic trail-distance values.

Output from this module consists of a minimum CDV grid for the park landscape and a summary of area by access class and hiking distance. The output grid contains a coded access value or the minimum CDV in km for every cell on the park landscape (Fig. 6). Coded values are used to indicate steep slopes (-1) and isolated areas (-2). The park_boundary layer is used to define the extent of the park landscape. CDVs are calculated and output only for park lands. Background (non-park area) is set to -9999. CDVs are limited to the size of a signed, short integer. If a CDV is >32768, the cell value is re-assigned to 32768, and a warning message appears on the computer screen. The resulting grid can be imported to ArcGIS with the asciigrid command. The percentage of park area by access class (too steep = -1, isolated = -2) and 1-km distance classes (where 1 = 0-1 km, 2 = >1-2 km, etc.) is tallied and output in tabular format.

Program Syntax. To operate travel_costsa, the program must be stored as travel_costsa.exe. To run this module, enter the following:

```
travel_costsa <rows> <columns> <spatial_grain_m> <slope_mask> <no. of iterations>
<parkboundary_layer> <road_layer> <min_road_code> <slope_layer> <aspect_layer> <trail-
distance_layer> <mode> <output_grid> <output_tally>
```

where;

<rows> is the number of rows in the data layers,
<columns> is the number of columns in the data layers,
<spatial_grain_m> is the spatial grain of the input layers (meters),
<slope_mask> is the percent slope that defines unsafe conditions,
<no. of iterations> see trail_roads_dista.exe
<parkboundary_layer> is the name of the park-boundary layer,
<roads_layer> is the road layer,
<min_road_code> indicates the minimum numeric code for roads in road_layer,
<slope_layer> is the name of the percent-slope layer,
<aspect_layer> is the name of the aspect layer,
<trail-distance_layer> is the name of the file generated by trail_roads_dista.exe,
<mode> is 1 for internally static trail-distance layer, 2 for internally dynamic trail-distance layer,
<output_grid> is the file name for the resulting ASCII-formatted grid containing the minimum slope-adjusted hiking distance (1-km classes) or an accessibility code for every landscape cell in a park,
<output_tally> is the file name for the tally of percentage of land area by access class and travel-distance category (-2 for isolated, -1 for steep slopes, else distance in 1-km intervals).

The program travel_costsa.exe can reside anywhere on your computer, but an explicit path to the directory containing the program must be inserted via the PATH command to execute the program from any directory. Alternatively, the path can be specified in the command-line syntax. E.g., h:\access\travel_costsa <> <> … .

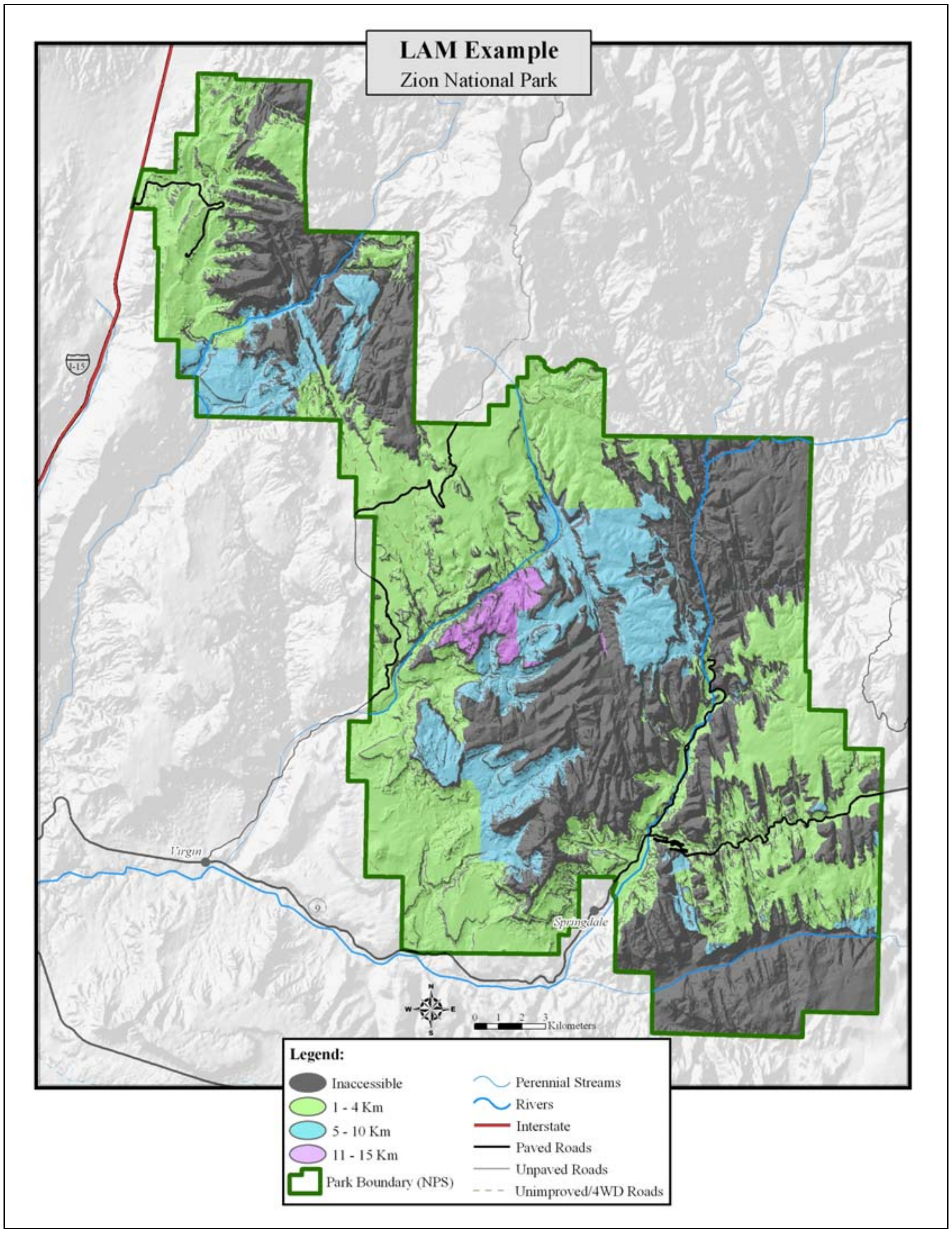


Figure 6. LAM results for ZION National Park. Inaccessible includes steep slopes and isolated areas. In this example, the 1-km distance values output by LAM are aggregated into three distance classes for visual clarity.

Key Assumptions and Limitations

1) Stream networks are not explicitly used as travel corridors. Certain streams may serve the same function as trails; that is, they intersect with roads (or trails) and provide hiking access to remote areas due to low or intermittent flow. Where applicable, specific stream networks can be included as trails in the trail layer used in trail_roads_dista.exe. Additionally, large or deep streams are not considered to inhibit travel across a landscape. Future enhancements are required to incorporate streams as barriers.

2) Roads that are not in the park, that do not abut the park boundary, and that do not intersect with trails that lead into the park are not used in deriving distance values in travel_costsa.exe. The underlying assumption for the above conditions is that the area between roads and park lands is not accessible. This assumption is valid where private lands are involved, but otherwise is fairly conservative. A less conservative approach will require ownership information for adjacent lands, and rule sets for determining access.

3) LAM assumes that all roads are equal and all trails are equal. LAM does not differentiate between 2wd, 4wd, unimproved, or paved roads. Similarly, trail quality is not differentiated in any way. The user is responsible for ensuring that the input road and trail layers are accurate and representative of roads and trails that can be accessed.

4) The amount of time or distance from a base camp to road segments intersecting a trail outside a park or to road segments inside a park are not included in travel costs estimates.

5) The minimum CDV grid output by travel_costsa.exe includes a distance value for road cells. This is simply one-half of the slope distance of a road cell. A more realistic approach may be to mask-out or recode road cells to avoid including road area as accessible landscape area.

6) The LAM modules do not contain safeguards for GIGO (garbage in, garbage out). The user is totally responsible for ensuring that data layers are representative and accurate.

LAM Revision History

Prev. Version #	Revision Date	Author	Changes Made	Reason for Change	New Version #
1.0 (4/04/05)	6/21/05	SLGarman – NCPN Ecologist	No code changes	Documentation modified to reflect the standard use of ASCII I/O versions, and edited for clarity.	1.0 (6/21/05)